An Object-oriented Synchronous Programming Language

Synchrony - Ten Years After

Reinhard Budde

Axel Poigné

Karl-Heinz Sylla

Overview

- Highlights
- Reactive Components
- Synchronous Styles
- Hybrid Systems
- Semantics
- Programming Environment
- Conclusions & Outlook

Highlights

- supports object-oriented design (using a Java-like language)
- integrates synchronous styles
- targets micro controllers

Fraunhofer Institut
Autonome Intelligente
Systeme

Reactive Components

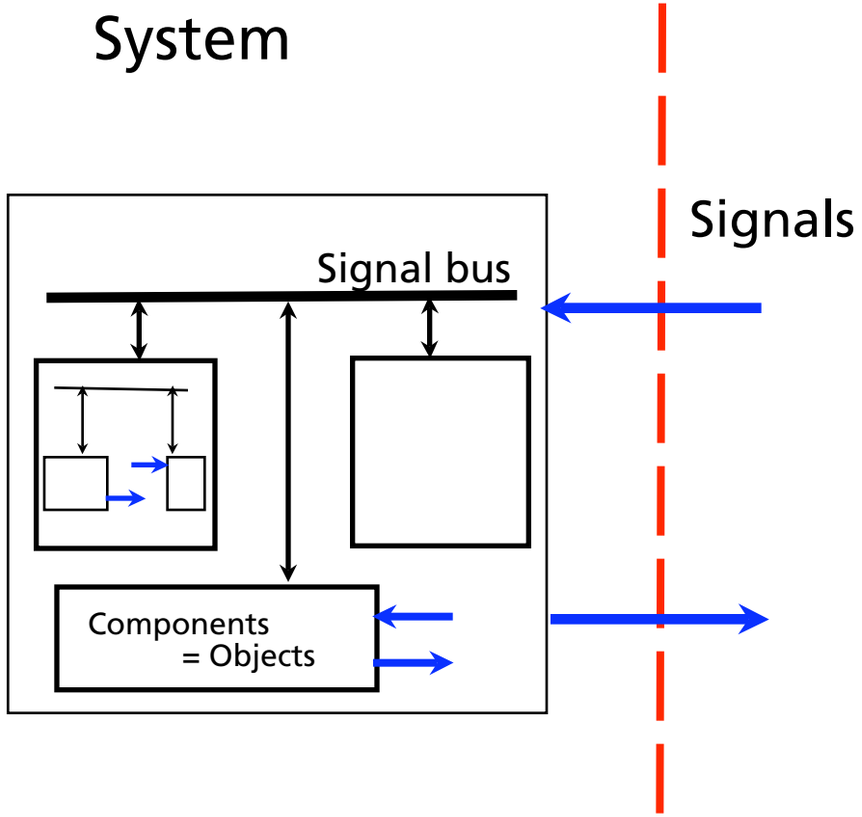
read only

interface to environment

```
class Basic {
   static final time timing = 250msec;

   Sensor button  = new Sensor(new SimInput());
   Signal red_led = new Signal(new SimOutput());

   public Basic () {
      active {
         loop {
            await ?button;
            emit  red_led;
            next;
         };
      };
   };

   public static void main (String[] args) {
      while (instant() == 0) {};
   };
}
```


Fraunhofer Institut Autonome Intelligente Systeme

Reactive Compents II



Fraunhofer Institut
Autonome Intelligente
Systeme

Synchronous Styles

- Imperative *(Esterel - like)*
- Data Flow *(Lustre - like)*
- Hierarchical State Machines *(Statecharts - like)*

Fraunhofer Institut
Autonome Intelligente
Systeme

Synchronous Styles: Imperative Style - Syntax

Types

Sensor, **Sensor**< *type* > *type = primitive, restricted reference*
Signal, **Signal**< *type* >
DelayedSignal, **DelayedSignal**<*type*>
time

Statements

emit s; , emit s(v);
assignment; *method_call*; *if reactive = module*
next;
await *expr*; **await** *time_expr*;
cancel { } when () else when …
activate { } when ();
loop { };
if ( ) { } else { };
[[|| ||]] *parallel*

Fraunhofer Institut
Autonome Intelligente
Systeme

Synchronous Styles: Imperative Style - New Features

Time Races

```
[[... data-op1 ...||... data-op2 ...]]
```

Restricted to classes
(attributes and methods are private)

Resolved using *precedence*

```
precedence {
  data-op1 < data-op2;
};
```


Fraunhofer Institut
Autonome Intelligente
Systeme

New Features Continued

Multiple Emittance

no combinators

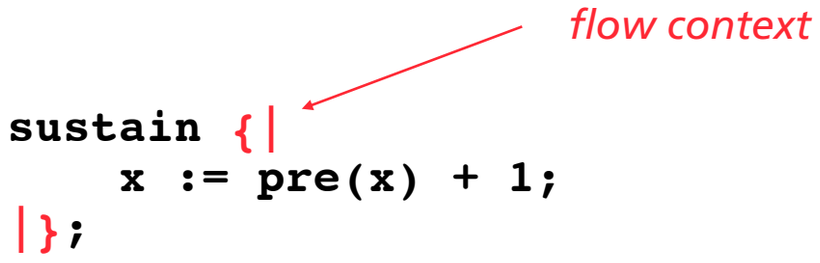
```
[[... emit s(v1); ...||... emit s(v2); ...]];
```

Either avoid or use *labels* and *precedence*

```
[[... l1::emit s(v1); ...||... l2::emit s(v2); ...]];

precedence {

  l1:: < l2:: ;

};
```


Fraunhofer Institut
Autonome Intelligente
Systeme

Synchronous Styles: Data Flow



- handled like any other statement (can be preempted)
- flow expressions with all the Lustre operator
- clocks

Fraunhofer AIS Institut Autonome Intelligente Systeme

Interrupt: Concerning Java

All compiled to C
(HW formats)

- ¬ *Dynamic Loading*
- ¬ *No packages (yet ?)*
- ¬ *Exception Handling only top level*
- ¬ *No Hiding (for good software)*
- ¬ *Only one- and two-dimensional arrays (for efficiency)*
- ¬ *Semicolon ";" is a must as terminator*

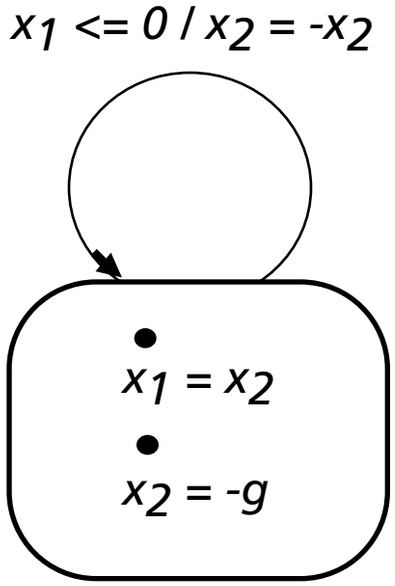
- \+ *more integer types: int8, unint8,..., uint64, int64*
- \+ *native C (constants and functions)*
- \+ *Parametrised Types "`List<T>`"*

Fraunhofer AIS Institut
Autonome Intelligente
Systeme

Hybrid Systems: Bouncing Ball

- $x_1 = x_2$ $\quad x_1 > 0$
- $x_2 = -g$

Change direction
if x1 <= 0



Fraunhofer Institut
Autonome Intelligente
Systeme

Bouncing Ball II

```
automaton {
   init { emit x1(height);
          emit x2(0.0);
          next state move; };

   state move
      during {| x1 := pre(x1) + x2*dt;
                x2 := -c*pre(x2) -> pre(x2) - g *dt;
      |}
   when ($x1 <= 0.0) { next state  move; };
};
```

change of direction

- **pre** *and* **->** *are defined relative to flow context*

Fraunhofer Institut
Autonome Intelligente
Systeme

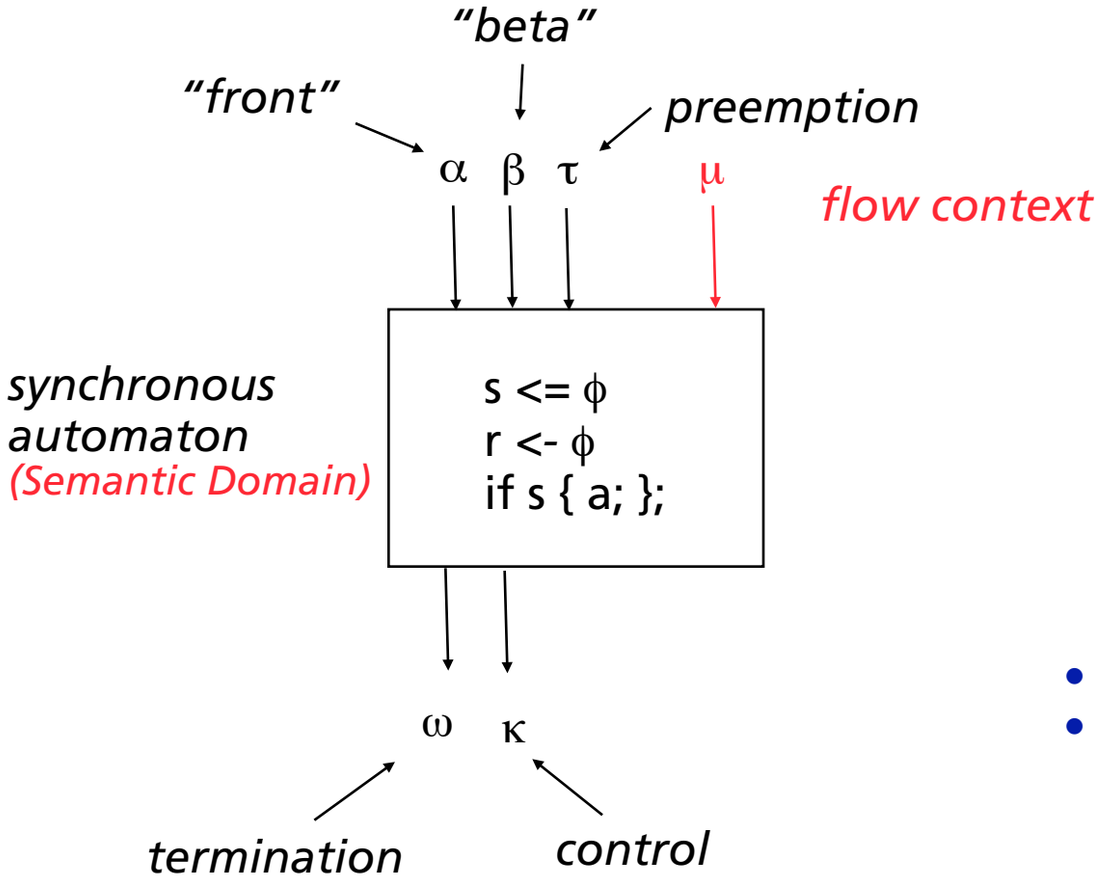
Semantics: Signals

- all Signals have a clock
- Signals are present if updated
- Signals may be updated using
  - emit Statement
  - flow equation: then clocks are checked
- Signals that are emitted are always on base clocks
- Clocks are part of a Signal type: **`Signal{`***clock***`}<`***type***`>`**

  `(Signal<`*type***`> = Signal{true}<`***type***`>)`**

Fraunhofer AIS Institut Autonome Intelligente Systeme

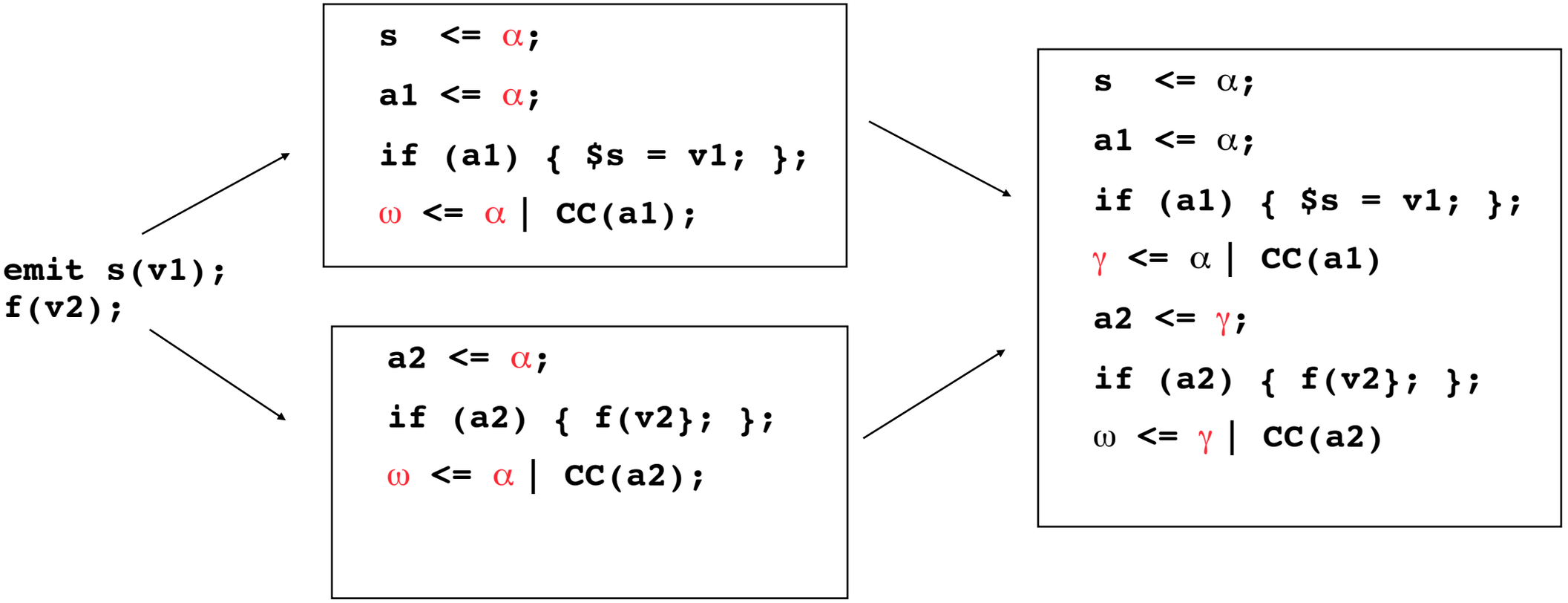
Semantics: Translation Scheme (Model of 93/98)



- *Semantics is "compositional"*
- *Realisation as denotational semantics in ocaml*

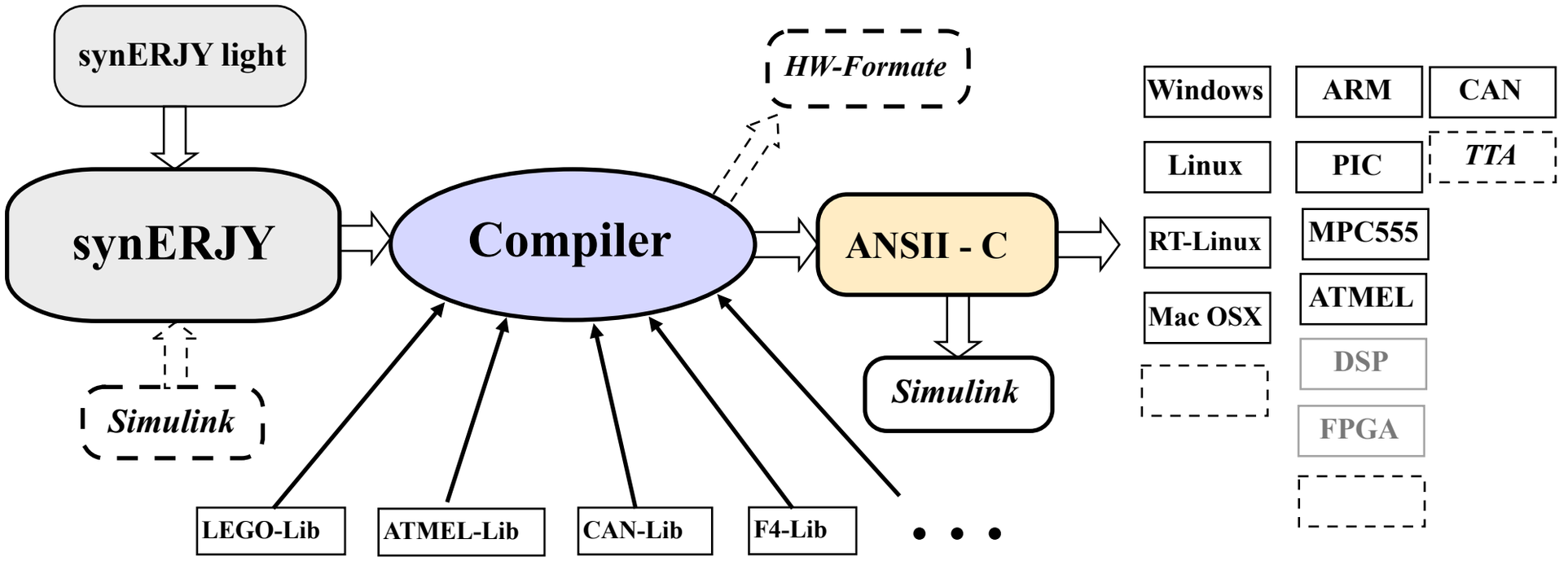
Fraunhofer Institut
Autonome Intelligente
Systeme

Translation Scheme Example



Fraunhofer Institut
Autonome Intelligente
Systeme

Programming Environment



Fraunhofer Institut
Autonome Intelligente
Systeme

Conclusions

- Language is stable / compiler reasonably mature
- Usage
  - Teaching
  - Robotics Applications
  - Project on smart materials
- Experience is somewhat annoying
  - people use only one style (imperative / state-machines), and
  - encode their standard programming style

Fraunhofer Institut
Autonome Intelligente
Systeme

Outlook

- allow several reactive engines (several calls of **`instant`**)
- Support for Digital Signal Processing
- Generate Code for DSP (more long term: FPGA)
- Optimisation of the compiler with regard to different targets
- Minor Points: add functional nodes, improve simulator, ...

Fraunhofer Institut
Autonome Intelligente
Systeme